BareCloud: Bare-metal Analysis-based Evasive Malware Detection

Dhilung Kirat
University of California, Santa Barbara
dhilung@cs.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
vigna@cs.ucsb.edu

Christopher Kruegel
University of California, Santa Barbara
chris@cs.ucsb.edu

Abstract

The volume and the sophistication of malware are continuously increasing and evolving. Automated dynamic malware analysis is a widely-adopted approach for detecting malicious software. However, many recent malware samples try to evade detection by identifying the presence of the analysis environment itself, and refraining from performing malicious actions. Because of the sophistication of the techniques used by the malware authors, so far the analysis and detection of evasive malware has been largely a manual process. One approach to automatic detection of these evasive malware samples is to execute the same sample in multiple analysis environments, and then compare its behaviors, in the assumption that a deviation in the behavior is evidence of an attempt to evade one or more analysis systems. For this reason, it is important to provide a reference system (often called bare-metal) in which the malware is analyzed without the use of any detectable component.

In this paper, we present BareCloud, an automated evasive malware detection system based on bare-metal dynamic malware analysis. Our bare-metal analysis system does not introduce any in-guest monitoring component into the malware execution platform. This makes our approach more transparent and robust against sophisticated evasion techniques. We compare the malware behavior observed in the bare-metal system with other popular malware analysis systems. We introduce a novel approach of hierarchical similarity-based malware behavior comparison to analyze the behavior of a sample in the various analysis systems. Our experiments show that our approach produces better evasion detection results compared to previous methods. BareCloud was able to automatically detect 5,835 evasive malware out of 110,005 recent samples.

1 Introduction

The malware threat landscape is continuously evolving. Early detection of these threats is a top priority for enterprises, governments, and end users. The widely-deployed signature-based and static-analysis-based detection approaches can be easily evaded by techniques commonly seen in the wild, such as obfuscation, polymorphism, and encryption. Therefore, dynamic malware analysis tools have recently become more popular to automate the analysis and detection of these threats [1, 14, 35]. These systems execute the suspicious sample in a controlled environment and observe its behavior to detect malicious intent. While this dynamic analysis approach is more effective against common static analysis evasion techniques, it faces a different set of challenges. More specifically, a malware sample, when executed, can detect the analysis environment and refuse to perform any malicious activity, for example by simply terminating or stalling the execution.

Malware authors have developed several ways to detect the presence of malware analysis systems. The most common approach is based on the fingerprinting of the runtime environment of the analysis system. This includes checking for specific artifacts, such as some specific registry keys, background processes, function hooks, or IP addresses that are specific to a known analysis tool. These artifacts must be known to the malware authors in advance to develop the corresponding fingerprinting techniques. Another approach leverages the fact that most of the analysis systems use emulated or virtualized environments as their malware execution platform. Such execution platforms can be detected by checking the platform-specific characteristics that are different with respect to a baseline environment (i.e., an unmodified operating system installed on real hardware, often referred to as a "bare-metal" installation). Such characteristics can be the timing properties of the execution, or a small variation in the CPU execution seman-

tics [31, 32].

Public-facing malware analysis systems are particularly vulnerable to the first approach to fingerprinting. This is because an attacker can submit malware samples specifically designed to extract the malware analysis environment artifacts to be then used in fingerprinting the analysis system. Private malware analysis systems are less prone to this type of fingerprinting. However, because of the internal sharing of malware samples among these private and public analysis systems, private systems may also be vulnerable to such fingerprinting [37].

One way to prevent the fingerprinting of the analysis environment is to construct a malware analysis system indistinguishable from a real host. Such systems are also known as *transparent* analysis systems. One of the first transparent analysis systems, called Cobra [34], tries to achieve this by developing a stealthy analysis environment using binary translation. However, this approach can only prevent known fingerprinting techniques. Ether [14] is a more robust transparent analysis system that leverages hardware virtualization to maintain the CPU execution semantics of a hardware CPU. However, the system introduces significant performance overhead when performing fine-grained monitoring, which is required to produce a comprehensive malware behavioral profile. With such performance overhead, it is fundamentally infeasible to make it transparent, especially if the malware execution has access to an external timing source [20].

Instead of preventing the fingerprinting of the analysis system, some of the recent works have focused on detecting a deviation of the malware behavior in different analysis environments [9, 13, 23, 24, 28]. The approach is to execute a malware sample in different analysis environments and compare their behavioral profiles to find a deviation. A behavioral profile is a higher-level abstraction of the activities performed by a malware sample when executed. The assumption is that the presence of such deviations is evidence of an attempt to fingerprint and evade one or more analysis systems. This is a generic and robust approach because it can detect evasion regardless of the knowledge of the techniques used by the malware sample in order to fingerprint and evade the analysis system. This approach assumes that the malware shows its malicious behavior in one of the analysis systems, also known as the *reference* system. However, all previous approaches have used emulated or virtualized environments for observing the deviation in the malware behavior, and such environments are known to be detectable. If all of the analysis systems are evaded by a malware sample, no significant deviation may be present in the execution traces. Moreover, some of the analysis systems use in-guest modules for behavior extraction, which further compromises the transparency of the analysis system.

A malware analysis system that is indistinguishable from a real host is a system that uses an unmodified operating system installation that runs on actual hardware (i.e., a bare-metal system). However, this approach faces several fundamental challenges. One of the important challenges is to efficiently restore the analysis system after every analysis run. Recently, a bare-metal-based malware analysis system, called BareBox [25], proposed an efficient system-restore technique. In this technique, the physical memory of the host is partitioned and only one partition is used for the analysis environment, while another partition is used for a snapshot of the system to be restored. Whenever needed, an external operating system located outside the physical memory of the analysis environment performs the restoration of the physical-memory snapshot, without the need for a reboot. However, a sophisticated malware can forcefully probe the physical memory and detect the presence of the BareBox system. Another bare-metal based malware analysis framework is Nvmtrace [5]. This system leverages IPMI (Intelligent Platform Management Interface) technology to automate the power cycle of the bare-metal analysis system. However, a complete reboot of the system is required after every analysis run. Another challenge to the bare-metal based malware analysis system is the extraction of the behavioral profile. To this end, no process-level behavior, such as process creation, termination, and hooking activities, can be extracted from a bare-metal analysis system without introducing some form of an in-guest analysis component. However, the presence of such components inside the system violates the transparency requirement and makes the system detectable. Because of this limitation, the observable malware behavior on a pure bare-metal system is limited to the disk-level and network-level activities. When only the disk-level and network-level behaviors are available, it may not be possible to perform an in-depth behavioral analysis, but these types of activity can be effectively used for detecting evasive behavior.

In this paper, we present BareCloud, a system for automatically detecting evasive malware. BareCloud detects evasive malware by executing them on a bare-metal system and comparing their behavior when executed on other emulation and virtualization-based analysis systems. Our bare-metal system has no in-guest monitoring component. This approach provides a robust transparent environment for our *reference system* where both user-mode and kernel-mode malware can be analyzed. BareCloud transparently extracts the behavioral profile of the malware from its disk-level and network-level activity. The disk-level activity is extracted by comparing the system's state after each malware execution with the initial *clean* state. Using the understanding of the operating system of the analysis host, BareCloud also extracts

operating-system-level changes, such as changes to specific registry keys and system files. Network-level activities are captured on the wire as a stream of network packets. This approach extracts malware behavior only from the persistent changes to the system. In principle, a malware sample could perform its activities without causing any persistent change, or could revert any changes after the activities are carried out. However, to perform any substantially malicious activity, a malware has to depend on some persistent change to the system, or it has to interact with external services, such as a C&C server. Both types of activities are transparently observable in our system.

When comparing the behavior of a malware sample on multiple platforms, previous works have considered the behavioral profiles purely as sets or bags of elements drawn from a flat domain, and computed their similarity using traditional set-intersection-based methods [8, 13, 28]. Set-intersection-based measures may not accurately capture similarity when data is sparse or when there are known relationships between elements within the sets [19]. For example, if two behavioral profiles under comparison contain a large number of similar file activities, but only one profile exhibits some network activities, set-intersection-based similarity measures, such as Jaccard similarity, produce a high similarity score, and fail to properly capture the lack of similarity among network activities. One may compute the similarity of the file activities and the network activities separately. However, similar problems exist; for example, two profiles may contain large number of similar DNS activities, but only one profile contains an HTTP request. It is important to identify such small-yet-important differences while comparing behavioral profiles for detecting evasions.

When manually comparing behavioral profiles, we start from generic questions such as "Do both profiles contain network and file activities?" If they do, we move on to other questions such as "Do these activities correspond to the same network or file objects?" This way of reasoning indicates that the behavioral profiles have an inherent similarity hierarchy based on the level of abstraction of the activities. Therefore, our similarity measure is based on the notion of the similarity hierarchy. Such hierarchy-base similarity can compute similarity at different levels of abstraction and identify activities that share similar characteristics even if they are not exactly the same. We show that this approach performs better than the set-intersection-based measure while comparing behavioral profiles for detecting evasive malware.

We compare the malware behavioral profile extracted from the bare-metal system with three major malware analysis platforms that are based on emulation and different types of virtualization, and we detect evasive behavior by detecting the deviation in the behavioral profile. Note that, beside evasion, there can be other factors that may cause a deviation in the behavioral profile. Section 4 describes how we mitigate those factors.

Our work makes the following contributions:

- We present BareCloud, a system for automatically detecting evasive malware. Our system performs malware analysis on a transparent bare-metal system with no in-guest monitoring component and on emulation-based and virtualization-based analysis systems.
- We introduce a novel evasion detection approach that leverages hierarchical similarity-based behavioral profile comparison. We show that this approach produces better results compared to the previous set-intersection-based approaches.
- We evaluate our system on a large dataset of recent real-world malware samples. BareCloud was able to detect 5,835 evasive malware instances out of 110,005 samples.

2 System Overview

The goal of our system is to automatically detect evasive malware by performing automated analysis of a large number of samples on a bare-metal reference system and other dynamic analysis systems. The goal is to identify deviations in the dynamic behavior of a sample when executed on different analysis environments. BareCloud achieve this by a multi-step process as depicted in Figure 1. The large volume of input samples is first pre-screened using the Anubis malware analysis framework [1]. The purpose of the pre-screening process is to select more interesting samples that are likely to have environment-sensitive behavior. These pre-screened samples are then executed on the cluster of bare-metal analysis hosts and on three other malware analysis systems, namely, Ether [14], Anubis [1], and Cuckoo Sandbox [2]. Each analysis system consists of multiple analysis hosts. The execution of the same sample in different systems is synchronized by the Scheduler component. Analysis hosts (workers) can independently join, perform analysis, and leave the BareCloud system. BareCloud extracts behavioral profiles from each of these analysis run, and, in the next step, it processes these profiles to detect evasive behavior.

3 Monitoring Environments

In this section, we describe the four malware analysis environments we use for monitoring the behavior of malware samples.

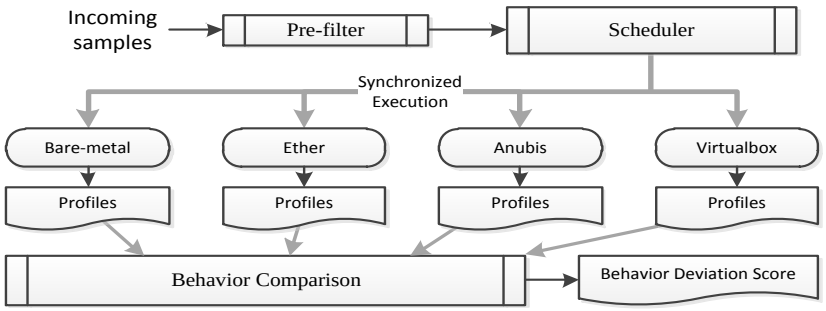

Figure 1: Overview of the system

3.1 Virtualization

We use Cuckoo Sandbox, which is based on VirtualBox [7], to provide a virtualization-based malware analysis platform. This is also known as a Type 2 hypervisor, that is, the virtualization host is a software application that runs on top of an operating system. Cuckoo Sandbox [2] supports the automation of the analysis process, and also includes function-hooking-based in-guest monitoring components. These components monitor execution-related events inside the analysis host. Several analysis and reporting plugins are available for specific needs. In this work, we use the analysis reporting plugin that includes Windows API call traces and network traffic. We use this trace to build the behavioral profile of a malware sample in the virtualized environment.

3.2 Emulation

We use the Anubis platform [1] to analyze malware in an emulated environment. Anubis is a whole-system emulation-based malware analysis platform. The emulator is based on Qemu [12]. No monitoring component is present inside the analysis environment other than some commonly used GUI automation tools. The emulator performs execution monitoring by observing the execution of pre-computed memory addresses. These memory addresses corresponds to important system API functions. Anubis is able to extract additional information about the API execution by inserting its own instructions to the emulator's instruction execution chain. Anubis implements a host of techniques, such as Product ID randomization, to prevent straightforward detection of the analysis system.

3.3 Hypervisor

We use Ether [14] to analyze malware in a hypervisor-based analysis environment. Ether is a Xen-based transparent malware analysis framework that utilizes Intel's VT hardware virtualization extensions [3]. The use of the hypervisor makes it possible to execute most of the malware instructions as native CPU instructions on the real hardware without any modifications. Thus, it does not suffer from inaccurate or incomplete system emulation issues that might affect emulation-based analysis systems. Ether can monitor a wide range of dynamic malware behaviors, such as system calls, memory writes, and fine-grained instructions execution. However, monitoring of memory writes and instruction-level trace introduces a substantial overhead and is only suitable for manual analysis. In this work, we only use Ether's coarse-grained system call trace collection capability. In addition, we also record all network communications during the malware execution. We combine the information from the system call trace and the network traffic to generate the behavioral profile.

3.4 Bare-metal

Our bare-metal malware analysis system is a cluster of hardware-based modular worker units. The workers' disks are based on a remote storage disk. This allows BareCloud to leverage copy-on-write techniques to perform disk restoration more efficiently when compared to a complete local disk overwrite. The bare-metal system also has a software-based remote control mechanism to automate the malware execution process on the workers. This mechanism is based on the IPMI remote administration features (e.g., IPMI allows to control the power cycle of the analysis worker units). We use the iSCSI protocol (Internet Small Computer System Interface) to attach remote disks to worker units. We used Logical Volume Manager (LVM)-based copy-on-write snapshots to host the remote disk used by the analysis system running on the worker units. After the completion of each malware analysis, the corresponding volume snapshot is recreated from a clean volume.

One of the critical components of a malware analy-

sis system is the *malware initiator*, which is the component that starts the execution of the malware. Usually, this component is implemented as some form of in-guest agent that waits for a malware sample through a network service. If the analysis system reboots after each analysis run, another approach can be to install a start-up entry in the system configuration that executes an application from a specific path. A malware sample can then be updated at this specific path for each analysis run by directly modifying the disk-image when the analysis system is offline. However, precise control over the malware execution duration is difficult when using this approach, as the overall execution time includes the system reboot time, which can vary among multiple reboots..

For a bare-metal analysis system, making its *malware initiator* component transparent is very important. This is because the malware can simply check for the presence of this component to fingerprint the environment. To this end, our system uses the network-based approach. The *malware initiator* removes itself and all of its artifacts after initiating the malware. This network-based approach also makes the malware execution duration more accurate, as it does not account for the reboot time.

3.5 User Environment

Apart from stock operating system, the environment installed inside the malware analysis systems includes some data and components that are usually present on a real host, such as saved credentials for common social networks, browser history, user document files, and other customizations. With this setup, we can observe additional malware behavior that we could not have observed using a bare user environment.

4 Behavior Comparison

In this section, we discuss malware behavioral deviation, behavioral profile extraction, and formalize behavioral profile comparison.

4.1 Behavior deviation

There are many factors that may cause a malware sample to show deviations in the dynamic behavior associated with different analysis environments. Hereinafter, we discuss each of these factors in detail.

- Evasive behavior of the malware sample:

  Deviation in the behavior may be the result of a successful fingerprinting of the analysis environment. This deviation is observable due to the change in the activities performed by the malware after the detection. This is the type of deviation we are interested in.

- Intrinsic non-determinism:

  A malware may have intrinsic non-determinism embedded in the code. That is, malware behavior might depend on some random value that it reads at the time of execution. For example, a malware sample may create a file with a random name. Randomization in the behavior can also result from the use of certain system services and APIs. For example, a successful call to `URLDownloadToFile` creates a random temporary folder to download the web content.

- Internal environment:

  Difference in the software environment of the different analysis systems may trigger different dynamic behaviors of the malware sample. For example, some malware may depend on a .NET framework installed in the analysis system, or may depend on the availability of a specific version of a system DLL. If one of the malware analysis environments does not contain such software components, the resulting malware behavior may be different.

- External environment:

  Another critical factor that may cause a deviation in the malware behavior is the external environment with which a malware sample can interact. In the context of malware execution, this external environment largely comprises of different network services, such as DNS and C&C servers. The non-deterministic nature of such network services may introduce deviations in the dynamic behavior of a malware sample. One simple way to minimize this factor is to completely disable access to external network environments. However, the network activity of a malware sample is one of the most important aspects to characterize the behavior of the sample. Hence, a successful behavior comparison of a malware sample requires the inclusion of its network activities.

Since our goal is to identify behavior deviations caused by the evasive technique employed by the malware sample, we need to minimize the effect of the three other factors that may cause a behavior deviation.

One approach to identifying intrinsic non-determinism is to execute the same sample in the same environment multiple times. By comparing the execution traces from

these different execution runs, non-deterministic behavior can be identified. Previous work [28] used this approach to filter out randomized behavior. However, this approach is resource- and time-expensive. Moreover, not all malware exhibit such randomized behavior.

In this work, we propose a more efficient hierarchical similarity-based approach to behavior comparison, described in Section 4.4. This approach is able to minimize the effect of intrinsic randomization without requiring multiple execution runs of the same sample in the same analysis environment. In order to address deviation caused by different internal environments, we must provide identical software environments to all analysis systems. Therefore, we prepared identical base software environments for all of our analysis systems.

Precisely controlling the behavior deviation introduced by the external environment is difficult. This is because these factors are not under our direct control. However, failure to minimize the impact of these factors may result erroneous behavior deviations. This consideration is important because most malware communicates with the external environment to carry out its malicious activities. To minimize the effect of the external environment, we implemented the following strategies.

- Synchronized execution: We execute the same malware sample in all analysis environments at the same time. The scheduler component facilitates the synchronization among different analysis hosts. By doing this, we minimize the behavior deviation that may be introduced by the variation of the external factors over time. For example, a malware may try to connect to a *fast-flux* network. The availability and the returned response of the C&C server and the DNS server may vary over time. If the malware is executed in different environments at different times, such variations in external environment may result in a spurious behavior deviation. Synchronized execution mitigates such differences.

- Identical local network: Malware can interact with the local network by different network-related activities, such as probing available local network services and accessing file shares. We expose all analysis systems to identical simulated local network environments.

- Network service filters: One approach to minimize the non-determinism introduced by different network services is to actively intercept network communications and maintain identical responses to identical queries among all instances of a malware running in different analysis environments. This requires an application-level understanding of the network services. To this end, we intercept all DNS and SMTP communications and respond with consistent replies in all analysis system. For example, the system responds with identical IP information to identical DNS queries coming from different analysis environments. With this setup, we are also able to sinkhole non-existent domain and SMTP communications to the local simulated network. This helps us observe more network behavior of a malware sample, which otherwise may not be observable.

4.2 Behavioral profile

After the execution of a malware sample in different analysis environments, we need to extract its behavioral profile for comparison. Usually, the behavioral profile is extracted from some form of dynamic execution trace, such as a system-call trace. Bayer et al. have introduced a comprehensive method of extracting behavioral profile from an augmented system-call trace. The additional information provides taint tracking of input and output parameters of system calls that provides dependency information between different system calls [10]. This approach has been used to cluster a large number of malware, and to compare malware behaviors [10, 28]. Similar approaches can be used in three of our analysis environments, where system-call traces are available. However, this system-call based approach is not directly applicable to our bare-metal malware analysis system, as we do not have access to the system-call trace.

Transient and resultant behavioral profile

A transient behavioral profile is a profile that represents all of the operations performed by a malware sample during its execution. The system-call-based behavioral profile discussed previously is a type of transient behavioral profile. This represents a more comprehensive view of *how* a malware performs its malicious activities. The resultant behavioral profile consists of the cumulative changes made by the malware from the beginning to the end of its execution. This includes those operations that make persistent changes to the system. Multiple similar operations to the same object are combined and represented as one operation to reflect the resulting effect of the operations. This represents a more summarized view of *what* a malware does to the system. A malware can obfuscate its transient behavior to evade transient behavior-base similarity detection. However, similar malicious activities produce similar resulting behavioral profiles, even if the transient behavior is obfuscated or randomized. This makes the comparison of malware behavior based on the resultant behavioral profile more robust.

The transparency requirement of our bare-metal analysis system limits us to the extraction of only the resulting behavioral profile. That is, the transient behaviors of process activities and filesystem activities are not available. However, we can extract the resulting filesystem behavior by comparing the disk contents from before and after the malware execution. Extraction of network behavior is straightforward using an external traffic capture component.

With these constraints in hand, we model our behavioral profile based on the model introduced by Bayer et al. [10], such that only the objects and the operations are used. That is, we take into consideration the object upon which a malware performs an operation that causes a persistent change to the object. Formally, a behavioral profile Π is defined as a 3-tuple.

$\Pi = (O, R, P)$

Where, O is the set of all objects, R is the set of all operations that causes persistent changes, and $P \subseteq (O \times R)$ is a relation assigning one or more operations to each object. Unlike in the model proposed in [10], where the objects and the operations are conceptualized as OS Objects and OS operations, we generalize the objects and operations to any environment entity with which a malware can interact. More details on objects and operations are provided hereinafter.

Objects

An object represents an entity, such as a file or a network endpoint, upon which a malware can perform some operation.

It is a tuple of `type` and `name` formally defined as follows.

$O = (obj\_type, obj\_name)$

$obj\_type ::= file|registry|sysconf|mbr|network$

The *file* type represents filesystem-specific file objects of the disk, the *registry* type represents registry keys, the *sysconf* type represents OS-specific system configurations, such as the boot configuration, *mbr* represents OS-independent Master Boot Record, and the *network* type represents network entities, such as a DNS server.

Operations

An operation generalizes the actions performed by a malware sample upon the above-described objects. An operation is formally defined as:

$R ::= (op\_name, op\_attribute)$

That is, an operation has a name and a corresponding attribute to provide additional information. As mentioned previously, only those operations that cause a persistent change to the system are included. For example, in case of a *file* type object, only the creation, deletion, and modification operations are included in the profile.

4.3 Behavior extraction

Our bare-metal system can only access the raw disk contents. We extract the filesystem behavior by comparing the filesystem state before and after the execution of a malware sample. A detailed understanding of the filesystem internal structures is required to extract such information. We leverage the functionalities provided by the SleuthKit framework [6] for extracting the file meta-data information from the raw disk image. By doing this, we are able to extract all file names in the disk, including some recently deleted files, along with their corresponding meta-data, such as size and modification date. We first build two sets representing the file object meta-data: the *clean* set and the *dirty* set, corresponding to the disk content before and after a malware execution. Extracting the deletion and creation operations of a file object are simple set operations. That is, any file not present in the dirty set is considered as deleted, and any file only present in the dirty set is considered as created. If a file is present in both sets with different meta-data, it is considered as modified. However, if a malware writes to a disk-sector (other than MBR) that is invisible to the filesystem, or modifies an existing file without changing the size and file-date meta-data, the current approach will not detect such changes. The straightforward way of comparing all file contents between two disk states can be very inefficient. This limitation can be mitigated by first detecting such changes in the disk sectors from copy-on-write data or iSCSI communication, and mapping the dirty sectors to files. Similar approach has been previously proposed [29]. To this end, we leave this improvement as a future work.

Registry behavior is extracted using a similar approach. We extract the meta-data of all the registry keys from the raw registry hive (registry database file) using the *registryfs* filesystem extension of the SleuthKit framework. Again, we build two sets representing the registry meta-data corresponding to the registry hive content before and after the malware execution. We perform set operations similar to the case of the filesystem to extract malware operations on the registry objects.

To extract the behavior of type *sysconf*, we process the filesystem and registry behavior to identify critical modifications to the system configuration. Some examples of the system configuration locations are listed in Table 1.

For the three other analysis systems, we process system-call traces to extract behavior information.

Table 1: Examples of the configuration locations

obj_type	obj_name	System path
sysconf	startup	HKLM/Software/Microsoft/Windows/CurrentVersion/Run
sysconf	startup	HKCU/Software/Microsoft/Windows/CurrentVersion/Run
sysconf	startup	HKLM/System/CurrentControlSet/Services
sysconf	boot	%SYSTEMROOT%/BOOT.INI
sysconf	autoexec	%SYSTEMROOT%/AUTOEXEC.BAT
sysconf	sysini	%SYSTEMROOT%/WINDOWS/SYSTEM.INI
sysconf	winini	%SYSTEMROOT%/WINDOWS/WIN.INI

Behavior normalization

Behavioral profile extracted from the difference of the initial and final disk states contains both malware behavior as well as the background operating system behavior. We need to filter out the features of the behavioral profile that do not correspond to the malware execution. One way to filter such features is to use the file modification timestamp of the file objects. That is, by selecting only those files that are created and modified during the time when the malware is executed, one can filter out unrelated file modifications that occur before and after the malware execution. However, some unrelated filesystem changes caused by the *base* operating system might still be present in the filtered profile. Moreover, many malware samples actively modify the system time, or tamper with the file meta-data to revert the file's modification time. Although the simple file time-stamp-based filter is efficient, this approach will fail in such situations.

Another approach to filter the background behavior is to first learn the behavioral profile of the *base* operating system and then filter this behavior from the profile generated by a malware execution. By doing this, we can overcome many of the shortcomings of the timestamp-based approach. This approach may exclude some malware operations that match the operation performed by the *base* operations. However, it is difficult to perform malicious actions using only operations that are also performed by the base operating system. Also, such operations are less important in defining the malicious behavior of the malware. We use this approach to filter our profiles. To extract the background behavior of the analysis system, we wrote a "void" program that does nothing other than stall infinitely. For each analysis environment, we extract the behavioral profile of the "void" program from all of its analysis hosts and combine them to build a generalized background profile. We use this profile to filter the behavioral profile of a malware execution.

Some objects used to describe the profile may be referenced using multiple names. For example, `\\?\C:\Documents and Settings` and `C:/DOCUME~1/` correspond to the same file object. We convert such identifiable object names to the same format. Different usernames may also result in different physical names for semantically similar file locations. For example, the locations `C:/DOCUME~1/USERA` and `C:/DOCUME~1/USERB` are semantically similar locations, which is the user's home directory. Some system APIs that create temporary files also generate different file paths, which are semantically similar. Many such temporary path names have known root locations and can be identified by their naming structure. We replace such occurrences in the object names with corresponding generic tokens.

4.4 Behavior comparison

Previous works have compared the persistent change-based behavioral profile using set-intersection-based methods over the feature set [13, 28]. However, when comparing behavioral profiles that only considers persistent changes, one can expect a sparse feature set. Furthermore, features within the profile are highly related and can be categorized in groups and subgroups. However, when the features are sparse or when there are known relationships between features within the set, set-intersection-based measures may not accurately capture the similarity [19].

Unlike previous works, we use a hierarchical similarity measure to overcome this problem. The hierarchy is associated with the different abstraction levels present in the behavioral profile. This approach makes our similarity measure less sensitive to randomization introduced by non-determinism in malware code. This is because the randomization is usually introduced only in one level of the hierarchy while keeping other levels of the hierarchy identical. For example, a malware may randomize the filename (obj_name) it creates, but perform the same *create* operation (op_name) on a *file* object (obj_type) with the same operation *attribute* ($op\_attribute$).

4.5 Hierarchical similarity

The notion of the hierarchical similarity is often used in text similarity, in mining association rules, and in various computer vision tasks for finding similar shapes [16, 17, 21]. We use a similar notion of hierarchical similarity to compare behavioral profiles. The similarity hierarchy of the behavioral profile is represented in Figure 2. As one can see, knowledge of the semantics and of the relationship between the objects is encoded in the representation. The leaves of the tree are the actual feature elements of the behavioral profile. The first level of similarity hierarchy is obj_type. An obj_type may have one or multiple obj_name, and each such obj_name can be associated with one or more op_name corresponding to various operations. Each such operation has one leaf node corresponding to the associated attribute of the operation. The leaf nodes are the feature elements whose attributes are represented by its parent nodes. For example, in the Figure 2, the ele-

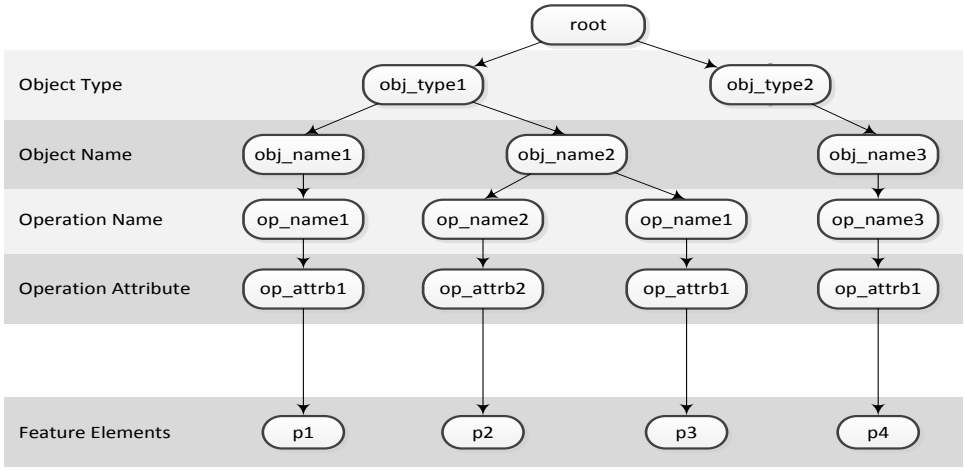

Figure 2: Behavior similarity hierarchy

ment p_1 is a feature element having the feature attributes $(obj\_type1, obj\_name1, op\_name1, op\_attrb1)$.

We compute the similarity in a two-step process. First we identify the matching nodes in the hierarchies of two behavioral profiles. We do this iteratively, starting from the first level (obj_type). For each of these matching nodes, we identify the matching nodes among their child nodes, i.e., the next level of hierarchy. We compute the similarity measure at each hierarchy level. Finally, we aggregate level similarity measures to compute the overall similarity.

The model

Let H be a rooted tree representing the similarity hierarchy, where all nodes have associated labels. For example, Figure 2 is an instance of H. Let L_H be all labels in H, and $L(H,d)$ be the set of labels of the nodes of H at depth d. Let δ be the height of the tree such that $L(H,\delta)$ is the set of all labels of the leaves of H. The set of labels $L(H,\delta)$ represents the feature elements p such that $p \in P$, where P represents the behavioral profile. In the example Figure 2, the leaf nodes p_1, p_2, p_3, and p_4 represents the feature elements of the behavioral profile. $L(H,\delta)$ and P are equivalent, one represented as leaves of the tree structure, another represented as a tuple of the feature attributes. With this, any P can be mapped into H. There is a hierarchy in $L(H,\delta)$, and hence in P, superimposed by H.

Let P_1 and P_2 be two behavioral profiles of a malware sample m from analysis systems a_1 and a_2. Let these behavioral profiles be mapped into hierarchies H_1 and H_2, instances of the hierarchical model H. Let $PL(H,l)$ be the label of the parent node of a node with label l, where $l \in L_H$, the set of all labels in H. Here, we want to find nodes with matching labels at each depth d whose parent nodes also have matching labels. We recursively define *match* and *candidate* for each level d as:

$$match_{H_1,H2}(d) = L(H_1,d) \cap L(H_2,d) \mid \forall l \in match_{H_1,H2}(d), PL(H_1,l) = PL(H_2,l) \text{ and } PL(H_1,l) \in match_{H_1,H2}(d-1) \quad (1)$$

$$candidate_{H_1,H2}(d) = L(H_1,d) \cup L(H_2,d) \mid \forall l \in candidate_{H_1,H2}(d), PL(H_1,l) \cup PL(H_2,l) \in match_{H_1,H2}(d-1) \quad (2)$$

where,

$$match_{H_1,H2}(0) = root. \quad (3)$$

We define $levelsim_{H_1,H_2}(d)$, the similarity of H_1 and H_2 at level d, as the Jaccard similarity coefficient. That is,

$$levelsim_{H_1,H_2}(d) = \frac{|match_{H_1,H2}(d)|}{|candidate_{H_1,H2}(d)|}. \quad (4)$$

We define the overall hierarchical similarity between behavioral profiles P_1 and P_2 as the arithmetic average of similarity at each level:

$$Sim(P_1,P_2) = \frac{1}{\delta}\sum_{d=1}^{\delta-1} levelsim_{H_1,H_2}(d). \quad (5)$$

This definition is consistent, since the right side of this equation always lies between 0 and 1. Hence, the *behavior distance* between P_1 and P_2 can simply be defined as:

$$Dist(P_1, P_2) = 1 - Sim(P_1, P_2). \quad (6)$$

This is possible because $Sim(P_1, P_2)$ is derived from the Jaccard similarity coefficients.

Finally, we define the behavior *deviation score* of a malware sample D among different analysis system $a_1 ... a_n$ with respect to the behavioral profile P_r extracted from the reference system a_r as the quadratic mean of the behavior distances as follows.

$$Deviation(D) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} Dist(P_r, P_i)^2}, \quad (7)$$

where n is the number of analysis systems, and P_i is the behavioral profile extracted from the analysis system a_i. This deviation score D is in interval [0,1], where the value of 0 means no deviation and the value of 1 means maximum deviation. We define a deviation threshold t. If the deviation score D exceeds this value, we consider the sample as evasive.

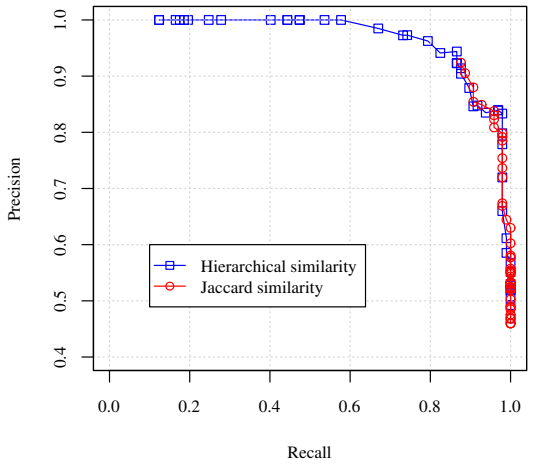

Figure 3: Precision-Recall analysis of the hierarchical similarity-based and the Jaccard similarity-based behavior comparison.

5 Evaluation

We evaluated our approach using Windows XP SP2 as the operating system for all analysis systems, as the weaker security measures of this operating system allow us to observe more malware behavior.

We conducted two experiments, which are detailed in the following sections.

5.1 Experiment I

In this experiment, we evaluated our hierarchical similarity-based behavior comparison approach.

Dataset

We first built the ground truth dataset of evasive malware and non-evasive malware samples. We received 234 recent and possibly evasive malware samples from a security company. We manually analyzed them and confirmed 111 samples from 29 families to be evasive (i.e., they fingerprint and evade at least one of the considered analysis environments). To build the dataset of non-evasive samples, we manually analyzed recent samples submitted to Anubis. By doing this, we selected 119 samples from 49 families that did not exhibit evasive behavior in any of the analysis environments.

We extracted the behavioral profiles of these samples from all analysis environments and computed the hierarchical similarity-based deviation score D with respect to the bare-metal analysis environment. We also computed the Jaccard distance-based deviation score as proposed in [28]. That is, the behavioral profile distance is computed as the Jaccard distance:

$$J(P_1, P_2) = 1 - \frac{P_1 \cap P_2}{P_1 \cup P_2}.$$

The Jaccard distance-based deviation score JD is computed as the maximum of the distances among different analysis systems as:

$$JD = \max_{1<i<n} \{ \max_{1<j<n, i \neq j} \{J(P_i, P_j)\}\}.$$

Since we do not extract multiple behavioral profiles of same malware sample from the same analysis environment, we did not compute the *intra-sandbox distance* as proposed in [28]. Instead, we wanted to compare the two different similarity approaches when the available behavioral profiles are limited to one profile per analysis environment.

We performed a precision-recall analysis by varying the threshold t for these deviation scores. If the deviation score is greater than t, the sample is considered evasive. The result is presented in Figure 3. We can clearly see that the hierarchical similarity-based deviation score gives better results. It is able to produce higher precision results while maintaining a high recall rate. While the highest precision the regular Jaccard distance-based similarity could produce is limited to 88.54%. This is because some non evasive samples have the same Jaccard distance-based deviation score as other evasive samples having highest deviation score. Hence, no matter how we vary the threshold t, precision does not increase.

Figure 4 shows the precision-recall characteristics of

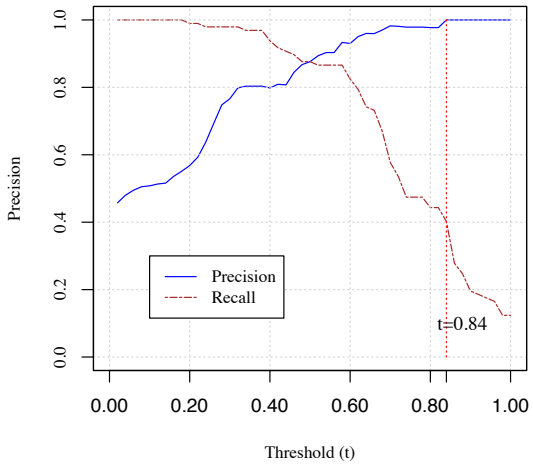

Figure 4: Precision-recall analysis of the behavior deviation threshold value t. Threshold value $t = 0.84$ gives the highest recall rate with 100% detection precision.

the hierarchical similarity-based deviation score with respect to different values for deviation threshold t. We can see that with the threshold value $t = 0.70$, more than half of the evasive samples can be detected with above 98% precision. While with threshold $t = 0.84$, we get 100% precision with the recall rate of 40.20%. In the next large-scale experiment, we used this threshold value $t = 0.84$ to have a high confidence on the detection results.

5.2 Experiment II

In this experiment, we applied our behavior comparison approach to the incoming malware feed received by Anubis. We first filtered the incoming samples based on the size and the type of the behavioral profile extracted in the Anubis analysis environment. This is required to select *interesting* samples from the large volume of the incoming malware without introducing any strong bias. This *pre-selection* process randomly selects samples from the following four groups.

- Samples with minimal activity: These are the samples that show minimal to no activity in the Anubis analysis environment (less than 1000 events). This group may contain evasive malware that successfully evade Anubis analysis environment.

- Samples with high system and network activity: These are the samples that show a substantial amount of system-related activity (more than 1000 events) and network activity in the Anubis analysis environment. We include this group to see if similar behavior can be observed in all other analysis environments.

- Samples with high network activity: These are the samples that show minimal system activity but high network activity (more than 10 packets) in the Anubis analysis environment.

- Samples with high system activity: These are the samples that show no network activity but high system activity in the Anubis analysis environment.

We selected 110,005 samples from the above groups observed during a four months period, starting from July 2013. We extracted behavioral profiles of these samples from all four analysis environments and computed the deviation scores. We used the behavior deviation threshold of $t = 0.84$ that was selected in the previous experiment. With this threshold, we were able to detected 5,835 samples as evasive. That is, these evasive samples evade one or more analysis environments. Since we do not have the ground truth for this dataset, we cannot provide the true precision of this detection.

We also categorized these evasive samples into environment-specific evasions by using the number of events observed within the various analysis environments. That is, if no activity that results in persistent change is observed in an analysis environment a_x while the same sample shows persistent change activities in at least one of the other analysis environments $[a_1, ..., a_n]$, we considered that as an evasion of the analysis environment a_x. The results are presented in Table 2. We found that 2,530 samples showed no activity that resulted in persistent change in all three common malware analysis environments, while such persistent change activities were observed in the bare-metal analysis environment. We see that many samples were able to evade the Ether environment, which is considered as a transparent environment. Manual analysis of those samples revealed that many samples were able to fingerprint Ether by leveraging the dependency of the Xen-hypervisor on Qemu-based *device models* to emulate required virtual devices, such as hard disks. Since Ether is based on Xen-hypervisor, it also requires the Qemu-based device models to run the unmodified guest, which acts as the transparent analysis host. Therefore, if a malware sample inside Ether tries to fingerprint its devices, it might identify Ether as a Qemu-based analysis system. In addition, system call tracing in Ether introduces additional overhead to the system, and, in some of the cases, regular timing-based attacks were successful in detecting the presence of the analysis system.

Table 2: Evasion detection on different environments

Environment	Detection count	Percentage
Anubis	4,947	84.78
Ether	4,562	78.18
VirtualBox	3,576	61.28
All	2,530	43.35
Total	5,835	

6 Limitations

In this section, we discuss the limitations of our approach.

One of the limitations inherent to all dynamic analysis systems is the inability of handling stalling code in a general way. A malware sample can wait for some time (e.g., tens of minutes) before performing any malicious activity. Kolbitsch et al. have proposed a technique to detect and mitigate malware stalling code [26]. However, the mitigation technique is only possible where fine-grained analysis is possible. The technique is not applicable in our bare-metal system. A more sophisticated stalling attack may encrypt the malicious payload with some weak encryption key, which the malware will brute-force during the execution. It is difficult to expedite such process to reveal the malicious activities within a limited execution time. To this end, BareCloud will not be able to observe activity in malware with stalling code.

Malware may employ heuristic checks for anomalies in the internal or external environments that are specific to malware analysis systems. For example, it can wait for real user activities, or try to resolve known non-existent domain using a public DNS server. If there is no user activity in the system or if every non-existent domain is resolved, it is likely that the system is an analysis environment. One of the mitigation techniques can be to remove external anomalies by not actively intercepting and modifying the network communication. User activities can be injected into the system by using automation techniques. However, transparently performing such automation in a bare-metal system is challenging. To this end, such heuristic checks will be able to detect our bare-metal analysis system.

Although, our similarity measure is less sensitive to randomization introduced by the non-deterministic code of a malware sample, high levels of randomization in the persistent changes will result in an erroneous detection of evasion. Other mitigation techniques, such as the multiple execution of the malware on the same analysis system, can be performed to mitigate this problem at the expense of more computational resources.

Both known and unknown fingerprinting techniques focused on detecting virtualized or emulated platforms will fail to detect BareCloud, because we are executing malware on a bare-metal hardware. However, there is the possibility that our system can be fingerprinted by examining unique software/hardware environment features, such as the MAC address of the network device or the presence of the iSCSI driver. In the case of emulated/virtualized environments, it is trivial to randomize such information for each malware analysis run. Since our system uses real hardware, introducing this randomization while preserving the transparency is difficult. The iSCSI driver detection can be mitigated by using more expensive hardware iSCSI initiator instead of a software iSCSI initiator. A hardware iSCSI initiator is a host bus adapter (HBA) that appears to the OS as a hardware storage device. To this end, our system runs as a private malware analysis framework and all outside network communications are blocked. A limited access to the Internet is provided through proxy. As long as the unique environment variables are not leaked to the malware authors, the system can be kept undetectable. However, a dedicated attacker may detect any dynamic analysis system that allows external network communications like ours by using active reconnaissance-based attacks [37]. Malware writers can upload decoy samples to public malware analysis systems so that it is eventually picked up by private analysis systems, such as ours. Such samples can leak unique environment artifacts of these analysis systems using "phoning home" technique and can be used for active reconnaissance.

7 Related works

7.1 Dynamic analysis

Researchers have developed many dynamic analysis tools to analyze malware. These tools mostly focus on extracting system call or Windows API call traces. Many of these analysis systems are based on sandboxing techniques [1, 4, 14, 35]. A sandbox is an instrumented execution environment that executes the malware sample in a contained way. Some of these sandboxes leverage in-guest techniques to trace Windows API calls, such as CWSandbox [35] and Norman Sandbox [4]. Other sandbox systems are implemented using emulation or virtualization technologies. VMScope [22], TTAnalyze [11], and Panorama [36] are some of the examples of emulation-based malware analysis systems. All of them are based on Qemu [12] and implement whole-system emulation. Other tools, such as Ether [14] and HyperDBG [15] are based on hardware-supported virtualization technology. While most system deal with user-land malware samples, some of the analysis systems are specifically targeted to analyze kernel-mode malware [27, 30].

7.2 Transparent analysis

Many transparent malware analysis systems have been proposed to defeat evasive malware. Cobra [34] was the first analysis system specifically focused on defeating anti-debugging techniques. However, Cobra runs its tool at the same privilege level as the malware. In principle, this approach makes it impossible to provide absolute transparency.

Many of the malware analysis tools based on the out-of-VM approach are designed to provide better transparency [1, 14, 22], as the analysis system is completely external to the execution environment. However, detection techniques have been developed to detect these analysis systems as well. There are several techniques to detect VMWare [9, 18, 33], as well as Bochs and Qemu [9, 18, 31]. Pek et al. [32] have shown that hardware virtualization-based Ether [14] can be detected using local timing attacks.

The most effective way to provide transparency is to run on real hardware, with an environment that has not been extended with analysis artifacts. BareBox [25] and Nvmtracer [5] both provide bare-metal environments for malware analysis.

7.3 Evasion detection

Chen et al. proposed a detailed taxonomy of evasion techniques used by malware against dynamic analysis system [13]. Lau et al. have employed a dynamic-static tracing technique to detect VM detection techniques. Kang et al. [24] proposed a scalable trace-matching algorithm to locate the point of execution diversion between two executions. The system is able to dynamically modify the execution of the whole-system emulator to defeat anti-emulation checks. Balzarotti et al. [9] proposed a system for detecting dynamic behavior deviation of malware by comparing behaviors between an instrumented environment and a reference host. The comparison method is based on the deterministic program execution replay. That is, the malware under analysis is first executed in a reference host while recording the interaction of the malware with the operating system. Later, the execution is replayed deterministically in a different analysis environment by providing system call return value recorded in the first run, in the assumption that any deviation in the execution is evidence of some kind of environment fingerprinting. Disarm [28] compares behavioral profiles of four emulation-based analysis environments. The behavior comparison requires each sample to be analyzed multiple times in each analysis environment. The main difference between Disarm and our work is that our analysis systems are based on four fundamentally different analysis platforms, including the transparent bare-metal environment with no monitoring component present in the hardware. Moreover, we propose an improved behavior comparison technique that captures the inherent similarity hierarchy of the behavior features, and do not require the resource-expensive execution of same sample multiple times in the same analysis environment.

7.4 Hierarchical Similarity

Hierarchies are used to encode domain knowledge about different levels of abstraction in the type of events observed. They have been used in different field of similarity detection, such as finding text similarity [16], detecting association rules using hierarchies of concepts [21], and finding similarity among deformable shapes [17]. Ganesan et al. [19] proposed a similarity measure that incorporates hierarchical domain structure. However, the similarity computation is focused on the element-level similarity rather than the profile-level similarity. It uses a modified version of cosine-similarity measure.

8 Conclusions

Dynamic analysis is an effective approach for analyzing and detecting malware that uses advanced packing and obfuscation techniques. However, evasive malware can fingerprint such analysis systems, and, as a result, stop the execution of any malicious activities. Most of the fingerprinting techniques exploit the fact that dynamic analysis systems are based on virtualized or emulated environments, which can be detected by several known methods. The ultimate way to thwart such detection is to analyze malware in a bare-metal environment.

In this work, we presented BareCloud, a system for automatically detecting evasive malware by using hierarchical similarity-based behavioral profile comparison. The profiles are collected by running a malware sample in bare-metal, virtualized, emulated, and hypervisor-based analysis environments.

Future work will focus on improving the transparency of the bare-metal analysis component and on developing an iSCSI module that can extract high-level, intermediate file system operation, providing a richer filesystem-level event trace.

9 Acknowledgments

This work is supported by the Office of Naval Research (ONR) under grant N00014-09-1-1042 and the Army Research Office (ARO) under grant W911NF-09-1-0553.

References

[1] Anubis. *http://anubis.iseclab.org*.

[2] Cuckoo Sandbox. *http://www.cuckoosandbox.org*.

[3] Intel Virtualization Technology. *http://intel.com/technology/virtualization*.

[4] Norman Sandbox. *http://www.norman.com/*.

[5] Nvmtrace. *http://code.google.com/p/nvmtrace*.

[6] SleuthKit. *http://www.sleuthkit.org*.

[7] VirtualBox. *http://www.virtualbox.org*.

[8] BAILEY, M., OBERHEIDE, J., ANDERSEN, J., MAO, Z. M., JAHANIAN, F., AND NAZARIO, J. Automated Classification and Analysis of Internet Malware. In *Symposium on Recent Advances in Intrusion Detection (RAID)* (2007).

[9] BALZAROTTI, D., COVA, M., KARLBERGER, C., KIRDA, E., KRUEGEL, C., AND VIGNA, G. Efficient Detection of Split Personalities in Malware. In *Symposium on Network and Distributed System Security (NDSS)* (February 2010).

[10] BAYER, U., COMPARETTI, P. M., HLAUSCHEK, C., KRUEGEL, C., AND KIRDA, E. Scalable, Behavior-Based Malware Clustering. In *Symposium on Network and Distributed System Security (NDSS)* (2009).

[11] BAYER, U., KRUEGEL, C., AND KIRDA, E. TTAnalyze : A Tool for Analyzing Malware. *European Institute for Computer Antivirus Research (EICAR)* (2006).

[12] BELLARD, F. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track* (2005).

[13] CHEN, X., ANDERSEN, J., MAO, Z. M., BAILEY, M., AND NAZARIO, J. Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware. In *IEEE Conference on Dependable Systems and Networks With FTCS and DCC* (2008), IEEE.

[14] DINABURG, A., ROYAL, P., SHARIF, M., AND LEE, W. Ether: Malware Analysis via Hardware Virtualization Extensions. In *ACM Conference on Computer and Communications Security (CCS)* (2008).

[15] FATTORI, A., PALEARI, R., MARTIGNONI, L., AND MONGA, M. Dynamic and Transparent Analysis of Commodity Production Systems. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2010), ACM.

[16] FELDMAN, R., AND DAGAN, I. Knowledge Discovery in Textual Databases (KDT). In *Conference on Knowledge Discovery and Data Mining (KDD)* (1995).

[17] FELZENSZWALB, P. F., AND SCHWARTZ, J. D. Hierarchical Matching of Deformable Shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2007), IEEE.

[18] FERRIE, P. Attacks on more virtual machine emulators. *Symantec Technology Exchange* (2007).

[19] GANESAN, P., GARCIA-MOLINA, H., AND WIDOM, J. Exploiting Hierarchical Domain Structure to Compute Similarity. *ACM Transactions on Information Systems (TOIS)* (2003).

[20] GARFINKEL, T., ADAMS, K., WARFIELD, A., AND FRANKLIN, J. Compatibility Is Not Transparency: VMM Detection Myths and Realities. In *USENIX Workshop on Hot Topics in Operating Systems (HotOS)* (2007).

[21] HAN, J., AND FU, Y. Discovery of Multiple-level Association Rules from Large Databases. In *Conference on Very Large Data Bases (VLDB)* (1995).

[22] JIANG, X., AND WANG, X. Out-of-the-Box Monitoring of VM-Based High-Interaction Honeypots. In *Symposium on Recent Advances in Intrusion Detection (RAID)* (2007).

[23] JOHNSON, N. M., CABALLERO, J., CHEN, K. Z., MCCAMANT, S., POOSANKAM, P., REYNAUD, D., AND SONG, D. Differential Slicing: Identifying Causal Execution Differences for Security Applications. In *IEEE Symposium on Security and Privacy* (2011).

[24] KANG, M., YIN, H., AND HANNA, S. Emulating Emulation-resistant Malware. *ACM Workshop on Virtual machine security* (2009).

[25] KIRAT, D., VIGNA, G., AND KRUEGEL, C. BareBox : Efficient Malware Analysis on Bare-Metal. In *Annual Computer Security Applications Conference (ACSAC)* (2011).

[26] KOLBITSCH, C., KIRDA, E., AND KRUEGEL, C. The Power of Procrastination: Detection and Mitigation of Execution-Stalling Malicious Code. In

ACM Conference on Computer and Communications Security (CCS) (2011).

[27] LANZI, A., SHARIF, M., AND LEE, W. K-Tracer: A System for Extracting Kernel Malware Behavior. In *Symposium on Network and Distributed System Security (NDSS)* (2009).

[28] LINDORFER, M., KOLBITSCH, C., AND COMPARETTI, P. M. Detecting Environment-Sensitive Malware. In *Symposium on Recent Advances in Intrusion Detection (RAID)* (2011).

[29] MANKIN, J., AND KAELI, D. Dione: A Flexible Disk Monitoring and Analysis Framework. *Research in Attacks, Intrusions, and Defenses* (2012).

[30] NEUGSCHWANDTNER, M., PLATZER, C., COMPARETTI, P. M., AND BAYER, U. dAnubis Dynamic Device Driver Analysis Based on Virtual Machine Introspection. *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2010).

[31] PALEARI, R., MARTIGNONI, L., ROGLIA, G. F., AND BRUSCHI, D. A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. In *USENIX Workshop on Offensive Technologies (WOOT)* (2009).

[32] PÉK, G., BENCSÁTH, B., AND BUTTYÁN, L. nEther: In-guest Detection of Out-of-the-guest Malware Analyzers. In *European Workshop on System Security (EUROSEC)* (2011), ACM.

[33] RUTKOWSKA, J. Red Pill or how to detect VMM using (almost) one CPU instruction. *http://invisiblethings.org/papers/redpill.html*, 2004.

[34] VASUDEVAN, A., AND YERRABALLI, R. Cobra: Fine-grained Malware Analysis using Stealth Localized-executions. In *IEEE Symposium on Security and Privacy* (2006).

[35] WILLEMS, C., HOLZ, T., AND FREILING, F. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy 5*, 2 (Mar. 2007).

[36] YIN, H., SONG, D., EGELE, M., KRUEGEL, C., AND KIRDA, E. Panorama : Capturing System-wide Information Flow for Malware Detection and Analysis. In *ACM Conference on Computer and Communications Security (CCS)* (2007).

[37] YOSHIOKA, K., HOSOBUCHI, Y., ORII, T., AND MATSUMOTO, T. Your Sandbox is Blinded: Impact of Decoy Injection to Public Malware Analysis Systems. *Journal of Information Processing* (2011).